Hindawi Publishing Corporation
Conference Papers in Science
Volume 2014, Article ID 906376, 8 pages
http://dx.doi.org/10.1155/2014/906376

Conference Paper

Using Feed Forward Neural Network to Solve Eigenvalue Problems

Luma N. M. Tawfiq and Othman M. Salih

Department of Mathematics, College of Education for Pure Science/Ibn Al-Haitham, Baghdad University, Baghdad, Iraq

Correspondence should be addressed to Othman M. Salih; oth_man2008@yahoo.com

Received 22 December 2013; Accepted 23 February 2014; Published 31 March 2014

Academic Editors: A. H. Bokhari and A. Jeribi

This Conference Paper is based on a presentation given by Luma N. M. Tawfiq at "The 3rd International Conference on Mathematical Sciences (ICMS3)" held from 17 December 2013 to 19 December 2013 in Kuala Lumpur, Malaysia.

The aim of this paper is to presents a parallel processor technique for solving eigenvalue problem for ordinary differential equations using artificial neural networks. The proposed network is trained by back propagation with different training algorithms quasi-Newton, Levenberg-Marquardt, and Bayesian Regulation. The next objective of this paper was to compare the performance of aforementioned algorithms with regard to predicting ability.

1. Introduction

These days every process is automated. A lot of mathematical procedures have been automated. There is a strong need of software that solves differential equations (DEs) as many problems in science and engineering are reduced to differential equations through the process of mathematical modeling. Although model equations based on physical laws can be constructed, analytical tools are frequently inadequate for the purpose of obtaining their closed form solution and usually numerical methods must be resorted to.

The application of neural networks for solving differential equations can be regarded as a mesh-free numerical method. It has been proved that feed forward neural networks with one hidden layer are capable of universal approximation, for problems of interpolation and approximation of scattered data.

2. Related Work

Neural networks have found application in many disciplines: neurosciences, mathematics, statistics, physics, computer science, and engineering. In the context of the numerical solution of differential equations, high-order derivatives are undesirable in general because they can introduce large approximation error. The use of higher order conventional Lagrange polynomials does not guarantee to yield a better quality (smoothness) of approximation. Many methods have been developed so far for solving differential equations; some of them produce a solution in the form of an array that contains the value of the solution at a selected group of points [1]. Others use basis functions to represent the solution in analytic form and transform the original problem usually to a system of algebraic equations [2]. Most of the previous study in solving differential equations using artificial neural network (Ann) is restricted to the case of solving the systems of algebraic equations which result from the discretisation of the domain [3]. Most of the previous works in solving differential equations using neural networks is restricted to the case of solving the linear systems of algebraic equations which result from the discretisation of the domain. The minimization of the networks energy function provides the solution to the system of equations [4]. Lagaris et al. [5] employed two networks: a multilayer perceptron and a radial basis function network to solve partial differential equations (PDE) with boundary conditions (Dirichlet or Neumann) defined on boundaries with the case of complex boundary geometry. Mc

Fall and Mahan [6] compared weight reuse for two existing methods of defining the network error function; weight reuse is shown to accelerate training of ODE; the second method outperforms the fails unpredictably when weight reuse is applied to accelerate solution of the diffusion equation. Tawfiq [7] proposed a radial basis function neural network (RBFNN) and Hopfield neural network (unsupervised training network) as a designer network to solve ODE and PDE and compared between them. Malek and Shekari Beidokhti [8] reported a novel hybrid method based on optimization techniques and neural networks methods for the solution of high order ODE which used three layered perceptron network. Akca et al. [9] discussed different approaches of using wavelets in the solution of boundary value problems (BVP) for ODE and also introduced convenient wavelet representations for the derivatives for certain functions and discussed wavelet network algorithm. Mc Fall [10] presented multilayer perceptron networks to solve BVP of PDE for arbitrary irregular domain where he used logsig. Transfer function in hidden layer and pureline in output layer and used gradient decent training algorithm; also, he used RBFNN for solving this problem and compared between them. Junaid et al. [11] used Ann with genetic training algorithm and log sigmoid function for solving first order ODE; Zahoor et al. [12] has been using an evolutionary technique for the solution of nonlinear Riccati differential equations of fractional order and the learning of the unknown parameters in neural network has been achieved with hybrid intelligent algorithms mainly based on genetic algorithm (GA). Abdul Samath et al. [13] suggested the solution of the matrix Riccati differential equation (MRDE) for nonlinear singular system using Ann. Ibraheem and Khalaf [14] proposed shooting neural networks algorithm for solving two-point second order BVP in ODEs which reduced the equation to the system of two equations of first order. Hoda and Nagla [4] described a numerical solution with neural networks for solving PDE, with mixed boundary conditions. Majidzadeh [15] suggested a new approach for reducing the inverse problem for a domain to an equivalent problem in a variational setting using radial basis functions neural network; also he used "cascade feed forward to solve two-dimensional Poisson equationwith back propagation and Levenberg-Marquardt train algorithm with the architecture three layers and 12 input nodes, 18 tansig. transfer function in hidden layer, and 3 linear nodes in output layer. Oraibi [16] designed feed forward neural networks (FFNN) for solving IVP of ODE. Ali [17] design fast FFNN to solve two-point BVP. This paper proposed FFNN to solve two-point singular boundary value problem (TPSBVP) with back propagation (BP) training algorithm. Tawfiq and Hussein [18] suggest multilayer FFNN to solve singular boundary value problems.

3. What Is Artificial Neural Network?

Ann is a simplified mathematical model of the human brain; it can be implemented by both electric elements and computer software. It is a parallel distributed processor with large numbers of connections; it is an information processing system that has certain performance characters in common with biological neural networks [19]. The arriving signals, called inputs, multiplied by the connection weights (adjusted) are first summed (combined) and then passed through a transfer function to produce the output for that neuron. The activation (transfer) function acts on the weighted sum of the neuron's inputs and the most commonly used transfer function is the sigmoid function (tansig) [17].

There are two main connection formulas (types): feedback (recurrent) and feed forward connection. Feedback is one type of connection where the output of one layer routes back to the input of a previous layer, or to same layer. Feed forward (FFNN) does not have a connection back from the output to the input neurons [20].

There are many different training algorithms, but the most often used is the Delta-rule or back propagation (BP) rule. A neural network is trained to map a set of input data by iterative adjustment of the weights. Information from inputs is fed forward through the network to optimize the weights between neurons. Optimization of the weights is made by backward propagation of the error during training phase.

The Ann reads the input and output values in the training data set and changes the value of the weighted links to reduce the difference between the predicted and target (observed) values. The error in prediction is minimized across many training cycles (iteration or epoch) until network reaches specified level of accuracy. A complete round of forward-backward passes and weight adjustments using all input-output pairs in the data set is called an epoch or iteration. If a network is left to train for too long, however, it will be overtrained and will lose the ability to generalize.

In this paper, we focused on the training situation known as supervised training, in which a set of input/output data patterns is available. Thus, the Ann has to be trained to produce the desired output according to the examples.

In order to perform a supervised training we need a way of evaluating the Ann output error between the actual and the expected output. A popular measure is the mean squared error (MSE) or root mean squared error (RMSE) [21].

4. Proposed Design

System design is the process of breaking a complex topic or substance into smaller parts to gain a better understanding of it. We try to design the EVP Solver using block diagrams.

The following are the actors of this application.

(1) End user: one who interacts with the system.

(2) System: receives commands and actions from the end user and performs required operations. FFNNs allow a conversion of a function from low-dimensional space to high-dimensional space (e.g., 1D–3D) in which the function will be expressed as a linear combination of ridge basis functions.

Provide the EVP of differential equation along with boundary conditions as input through GUI. Based on the EVP generates the data points. Determine the centers with respect to the generated data points. The data points and

eigenvalue should be within the solution space. If the data points and eigenvalue are out of the solution space then change the boundary conditions and again find out the data points and eigenvalue.

5. Description of the Method

In the proposed approach the model function is expressed as the sum of two terms: the first term satisfies the boundary conditions (BC) and contains no adjustable parameters. The second term can be found by using FFNN which is trained so as to satisfy the differential equation and such technique called collocation neural network.

In this section we will illustrate how our approach can be used to find the approximate solution of the general form a 2nd order EVP:

$$y''(x) = F\left(x, y(x), y'(x), \lambda\right), \tag{1}$$

where a subject to certain BC's and $x \in R$, $D \subset R$ denotes the domain, and $y(x)$ is the solution to be computed.

If $y_t(x, p)$ denotes a trial solution with adjustable parameters p, the problem is transformed to a discretized form:

$$\min_{p} \sum_{\vec{x}_i \in \widehat{D}} F\left(x_i, y_t(x_i, p), y_t'(x_i, p), \lambda\right) \tag{2}$$

subject to the constraints imposed by the BC's.

In our proposed approach, the trial solution y_t employs a FFNN and the parameters p correspond to the weights and biases of the neural architecture. We choose a form for the trial function $y_t(x)$ such that it satisfies the BC's. This is achieved by writing it as a sum of two terms:

$$y_t(x_i, p) = A(x) + G(x, N(x, p)), \tag{3}$$

where $N(x, p)$ is a single-output FFNN with parameters p and n input units fed with the input vector x. The term $A(x)$ contains no adjustable parameters and satisfies the BC's. The second term G is constructed so as not to contribute to the BC's, since $y_t(x)$ satisfy them. This term can be formed by using a FFNN whose weights and biases are to be adjusted in order to deal with the minimization problem.

6. Computation of the Gradient

An efficient minimization of (2) can be considered as a procedure of training the FFNN, where the error corresponding to each input x_i is the value $E(x_i)$ which has to be forced near zero. Computation of this error value involves not only the FFNN output but also the derivatives of the output with respect to any of its inputs.

Therefore, in computing the gradient of the error with respect to the network weights consider a multilayer FFNN with n input units (where n is the dimensions of the domain), one hidden layer with H sigmoid units, and a linear output unit.

For a given input x the output of the FFNN is

$$N = \sum_{i=1}^{H} v_i \sigma(z_i), \quad \text{where } z_i = \sum_{j=1}^{n} w_{ij} x_j + b_i. \tag{4}$$

denotes the weight connecting the input unit j to the hidden unit i, v_i denotes the weight connecting the hidden unit i to the output unit, b_i denotes the bias of hidden unit i, and $\sigma(z)$ is the sigmoid transfer function (tansig).

The gradient of FFNN, with respect to the parameters of the FFNN, can be easily obtained as

$$\begin{aligned} \frac{\partial N}{\partial v_i} &= \sigma(z_i), \\ \frac{\partial N}{\partial b_i} &= v_i \sigma'(z_i), \\ \frac{\partial N}{\partial w_{ij}} &= v_i \sigma'(z_i) x_j. \end{aligned} \tag{5}$$

Once the derivative of the error with respect to the network parameters has been defined, then it is straightforward to employ any minimization technique. It must also be noted that the batch mode of weight updates may be employed.

7. Illustration of the Method

In this section we describe solution of EVP using FFNN. To illustrate the method, we will consider the 2nd order EVP:

$$\frac{d^2 y(x)}{dx^2} = f\left(x, y, y', \lambda\right), \tag{6}$$

where $x \in [a, b]$ and the BC: $y(a) = A$, $y(b) = B$ (Dirichlet case) or $y'(a) = A$, $y'(b) = B$ (Neumann case) or $y'(a) = A$, $y(b) = B$ (Mixed case). A trial solution can be written as

$$\begin{aligned} y_t(x, p) = {} & \frac{(bA - aB)}{(b - a)} + \frac{(B - A)x}{(b - a)} \\ & + (x - a)(x - b) N(x, p), \end{aligned} \tag{7}$$

where $N(x, p)$ is the output of a FFNN with one input unit for x and weights p.

Note. $y_t(x)$ satisfies the BC by construction. The error quantity to be minimized is given by

$$E[p] = \sum_{i=1}^{n} \left\{ \frac{d^2 y_t(x_i, p)}{dx^2} - f\left(\frac{x_i, y_t(x_i, p), dy_t(x_i, p)}{dx} \right) \right\}^2, \tag{8}$$

Table 1: Analytic and neural solution of Example 1.

Input	Analytic solution	Output of suggested FFNN $y_t(x)$ for different training algorithms		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	1.000000000000000	1.000000000347198	1.000000000000027
0.1	0.223130160148430	0.223130160148430	0.223130160475769	0.223130160148043
0.2	0.002478752176666	0.002478752176666	0.002478752608702	0.002478752184984
0.3	$1.370959086384080e-06$	$1.370959086255397e-06$	$2.138822309394328e-05$	$1.370752687690491e-06$
0.4	$3.775134544279084e-11$	$-1.423704123482139e-07$	$9.137549916715670e-08$	$2.862885595256159e-09$
0.5	$5.175555005801869e-17$	$2.220446049250313e-16$	$-4.773932360535582e-09$	$-1.930348370038360e-08$
0.6	$3.532628572200807e-24$	$4.440892098500626e-16$	$1.076577582637128e-08$	$6.462875423718373e-08$
0.7	$1.200481799513899e-32$	$-2.716475933084439e-10$	$-6.335612035002214e-09$	$-9.347123508529620e-08$
0.8	$2.031092662734782e-42$	$-1.559072870804812e-10$	$-2.091692152816904e-08$	$2.220824357745954e-08$
0.9	$1.710883542651365e-53$	$7.175095973164410e-66$	$6.814833142243515e-10$	$6.371296168428131e-08$
1.0	$7.175095973164410e-66$	$1.589879339292111e-10$	$8.929697603576870e-08$	$-4.040193102294865e-08$

where the $x_i \in [a, b]$. Since

$$\frac{dy_t(x,p)}{dx} = \frac{(B-A)}{(b-a)} + \{(x-a)+(x-b)\} N(x,p) + (x-a)(x-b)\frac{dN(x,\vec{p})}{dx},$$
$$\frac{d^2 y_t(x,p)}{dx^2} = 2N(x,p) + 2\{(x-a)+(x-b)\} \times \frac{dN(x,\vec{p})}{dx} + \frac{(x-a)(x-b)\,d^2N(x,p)}{dx^2} \quad (9)$$

it is straightforward to compute the gradient of the error with respect to the parameters p using (5). The same holds for all subsequent model problems.

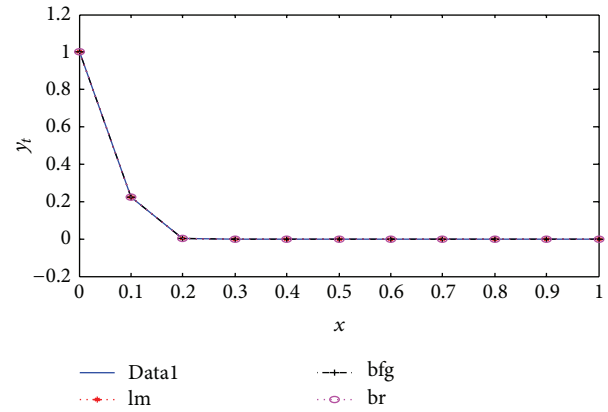

Figure 1: Analytic and neural solution of Example 1 using trainbfg, trainbr, and trainlm training algorithm.

8. Examples

In this section we report numerical result, using a multilayer FFNN having one hidden layer with 5 hidden units (neurons) and one linear output unit. The sigmoid activation of each hidden unit is tansig; the analytic solution $y_a(x)$ was known in advance. Therefore we test the accuracy of the obtained solutions by computing the deviation:

$$\Delta y(x) = \left| y_t(x) - y_a(x) \right|. \quad (10)$$

In order to illustrate the characteristics of the solutions provided by the neural network method, we provide figures displaying the corresponding deviation $\Delta y(x)$ both at the few points (training points) that were used for training and at many other points (test points) of the domain of equation. The latter kind of figures is of major importance since they show the interpolation capabilities of the neural solution which is to be superior compared to other solutions obtained by using other methods. Moreover, we can consider points outside the training interval in order to obtain an estimate of the extrapolation performance of the obtained numerical solution.

Example 1. Consider the following 2nd order EVP:

$$y'' + 2\lambda y' + 2y = 0, \quad 0 \le x \le 1. \quad (11)$$

With BC (Dirishlit case), $y(0) = 1$, $y(1) = e^{-\lambda}$.

The analytic solution is $y_a(x) = e^{-\lambda x^2}$; according to (8) the trial neural form of the solution is taken to be

$$y_t(x) = 1 + \left(e^{-\lambda} - 1\right)x + x(x-1)N(x,p). \quad (12)$$

The FFNN trained using a grid of ten equidistant points in $[0, 1]$ gave $\lambda = 150$; Figure 1 displays the analytic and neural solutions with different training algorithms. The neural results with different types of training algorithm such as Levenberg-Marquardt (trainlm), quasi-Newton (trainbfg), and Bayesian Regulation (trainbr) are introduced in Table 1 and its errors are given in Table 2; Table 3 gives the performance of the train with epoch and time and Table 4 gives the weight and bias of the designer network.

Table 2: Accuracy of solutions for Example 1.

The error $E(x) = \lvert y_t(x) - y_a(x)\rvert$, where $y_t(x)$ is computed by the following training algorithms		
Trainlm	Trainbfg	Trainbr
$2.220446049250313e-16$	$3.471978260449760e-10$	$2.708944180085382e-14$
$3.330669073875470e-16$	$3.273391557812033e-10$	$3.868017017794045e-13$
$1.314053033052431e-16$	$4.320360939662205e-10$	$8.317256171980203e-12$
$1.286829394127678e-16$	$2.001726400755920e-05$	$2.063986935888358e-10$
$1.424081636936567e-07$	$9.133774782171392e-08$	$2.825134249813369e-09$
$1.702890548670126e-16$	$4.773932412291132e-09$	$1.930348375213914e-08$
$4.440892063174340e-16$	$1.076577582637128e-08$	$6.462875423718373e-08$
$2.716475933084439e-10$	$6.335612035002214e-09$	$9.347123508529620e-08$
$1.559072870804812e-10$	$2.091692152816904e-08$	$2.220824357745954e-08$
$1.710883542650648e-53$	$6.814833142243515e-10$	$6.371296168428131e-08$
$1.589879339292111e-10$	$8.929697603576870e-08$	$4.040193102294865e-08$

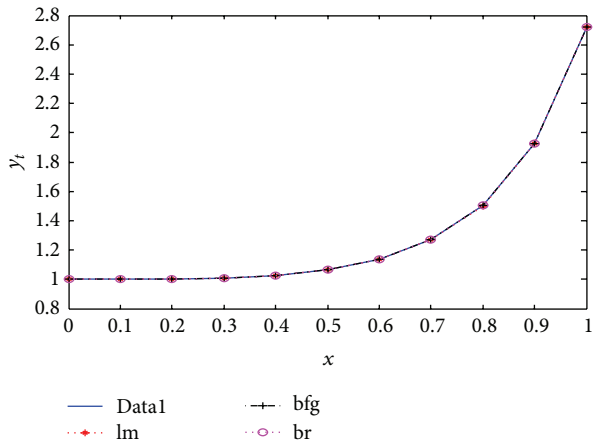

Figure 2: Analytic and neural solution of Example 2 using trainbfg, trainbr, and trainlm training algorithm.

Table 3: The performance of the train with epoch and time.

Train function	Performance of train	Epoch	Time	Msereg.
Trainlm	$4.56e-33$	6696	0:00:49	$1.6593e-015$
Trainbfg	$9.10e-19$	2297	0:00:25	$3.2785e-011$
Trainbr	$1.95e-14$	38445	0:05:25	$1.5938e-015$

Example 2. Consider the following 2nd order EBVP:

$$\left(x^{\lambda} y'\right)' = 4x^{\lambda+2}\left(3 + \lambda + 4x^4\right) y, \quad 0 \le x \le 1 \qquad (13)$$

with BC (Dirishlit case), $y(0) = 1$, $y(1) = e^1$.

The analytic solution is $y_a(x) = e^{x^4}$; according to (7) the trial neural form of the solution is

$$y_t(x) = 1 + \left(e^1 - 1\right) x + x(x-1) N(x, p). \qquad (14)$$

The FFNN trained using a grid of ten equidistant points in $[0, 1]$ gave $\lambda = 0.5$. Figure 2 displays the analytic and neural solutions with different training algorithms. The neural results with different types of training algorithm such as Levenberg-Marquardt (trainlm), quasi-Newton (trainbfg), and Bayesian Regulation (trainbr) are introduced in Table 5 and its errors are given in Table 6; Table 7 gives the performance of the train with epoch and time and Table 8 gives the weight and bias of the designer network.

Table 4: Weight and bias of the network for different training algorithms.

(a)

Weights and bias for trainlm		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.0092	0.2088	0.1319
0.4768	0.5205	0.9547
0.2503	0.2255	0.1239
0.3079	0.5672	0.1862
0.9669	0.9982	0.6465

(b)

Weights and bias for trainbfg		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.2691	0.9831	0.6981
0.4228	0.3015	0.6665
0.5479	0.7011	0.1781
0.9427	0.6663	0.1280
0.4177	0.5391	0.9991

(c)

Weights and bias for trainbr		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.5211	0.3955	0.9133
0.2316	0.3674	0.7962
0.4889	0.9880	0.0987
0.6241	0.0377	0.2619
0.6791	0.8852	0.3354

Table 5: Analytic and neural solution of Example 2.

Input	Analytic solution	Output of suggested FFNN $y_t(x)$ for different training algorithms		
x	$y_a(x)$	Trainlm	Trainbfg	Trainbr
0.0	1	1.000000000000000	0.999999791497422	1.000000531336361
0.1	1.000100005000167	1.000144087759814	1.000100368418017	1.000097184294931
0.2	1.001601280682940	1.001601280682940	1.001498511767109	1.001607051239879
0.3	1.008132893753152	1.008132893753152	1.008132867669427	1.008128167440348
0.4	1.025930494190382	1.025930494190382	1.026019395339251	1.025927732110325
0.5	1.064494458917860	1.064590790583371	1.064494437575892	1.064506007546008
0.6	1.138372943065416	1.138372943065416	1.138272638536007	1.138359307576485
0.7	1.271376281592894	1.269464922443141	1.271376301269824	1.271384988973427
0.8	1.506215178528160	1.500405060697450	1.506214213098092	1.506211973532767
0.9	1.927261339283878	1.927261339283878	1.927192081328471	1.927261991541353
1.0	2.718281828459046	2.718281828459046	2.718280402057775	2.718281767743469

Table 6: Accuracy of solutions for Example 2.

The error $E(x) = \lvert y_t(x) - y_a(x)\rvert$, where $y_t(x)$ is computed by the following training algorithms		
Trainlm	Trainbfg	Trainbr
$2.220446049250313e-16$	$2.085025777587291e-07$	$5.313363611314514e-07$
$4.408275964773445e-05$	$3.634178504796637e-07$	$2.820705235384580e-06$
$2.220446049250313e-16$	$1.027689158308309e-04$	$5.770556939177496e-06$
0	$2.608372495771505e-08$	$4.726312804681498e-06$
0	$8.890114886850320e-05$	$2.762080057561178e-06$
$9.633166551159533e-05$	$2.134196797065613e-08$	$1.154862814822799e-05$
0	$1.003045294094562e-04$	$1.363548893107414e-05$
0.001911359149753	$1.967692941917676e-08$	$8.707380532602116e-06$
0.005810117830711	$9.654300683337170e-07$	$3.204995393302212e-06$
$2.220446049250313e-16$	$6.925795540735358e-05$	$6.522574744760590e-07$
0	$1.426401270432365e-06$	$6.071557701048391e-08$

Table 7: The performance of the train with epoch and time.

Train function	Performance of train	Epoch	Time	Msereg.
Trainlm	$2.11e-32$	137	0:00:01	$3.0618e-006$
Trainbfg	$8.87e-23$	553	0:00:15	$2.7266e-009$
Trainbr	$4.77e-10$	3500	0:00:27	$3.9054e-011$

Rasheed [22] solved this problem using semianalytic technique and the results are given in Table 9.

9. Conclusion

From the above problems it is clear that the network which is proposed can handle effectively EVP and provide accurate approximate solution throughout the whole domain and not only at the training points. As evident from the tables, the results of proposed network are more precise as compared to method suggested in [22].

In general, the practical results for FFNN show the network which contain up to a few hundred weights with the Levenberg-Marquardt training algorithm (trainlm) having the fastest convergence than the network with trainbfg training algorithm and then the network with trainbr training

Table 8: Weight and bias of the network for different training algorithms.

(a)

Weights and bias for trainlm		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.5459	0.7896	0.4897
0.9432	0.7992	0.9729
0.3215	0.0496	0.7485
0.8065	0.2832	0.5678
0.6014	0.6535	0.2990

(b)

Weights and bias for trainbfg		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.8859	0.4737	0.8309
0.2138	0.9512	0.8246
0.0346	0.2490	0.4530
0.4511	0.3864	0.3806
0.0138	0.4314	0.9259

(c)

Weights and bias for trainbr		
Net.IW{1, 1}	Net.LW{2, 1}	Net.B{1}
0.3182	0.1756	0.9988
0.8993	0.5073	0.3745
0.0070	0.2938	0.5081
0.1739	0.8733	0.7485
0.7677	0.8235	0.3999

Table 9: The results of Example 2 given in [22].

x_i	Exact solution	Solution in [22] P_{21}	Error $\lvert y - P_{21} \rvert$
0	1.000000000000000	1.000000000000000	0
0.1	1.000100005000167	1.000100005056761	$5.659384072487228e^{-011}$
0.2	1.001601280682940	1.001601314236604	$3.355366451351927e^{-008}$
0.3	1.008132893753152	1.008133600590699	$7.068375460494991e^{-008}$
0.4	1.025930494190382	1.025933732312788	$3.238122405324617e^{-006}$
0.5	1.064494458917860	1.064499725618846	$5.266700985995243e^{-006}$
0.6	1.138372943065416	1.138376181379454	$3.238314038034318e^{-006}$
0.7	1.271376281592894	1.271376594860263	$3.132673687122889e^{-007}$
0.8	1.506215178528160	1.506214797396031	$3.811321289681757e^{-007}$
0.9	1.927261339283878	1.927261144233841	$1.950500372327468e^{-007}$
1	2.718281828459046	2.718281828459046	0
Max. error		$7.068375460494991e^{-008}$	
S.S.E		$4.949244162584533e^{-011}$	

algorithm. However, "trainbr" does not perform well for function approximation on problems. The performance of the various algorithms can be affected by the accuracy required of the approximation.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] K. M. Mohammed, *On solution of two point second order boundary value problems by using semi-analytic method [M.S. thesis]*, University of Baghdad, College of Education-Ibn-Al-Haitham, Baghdad, Iraq, 2009.

[2] R. J. LeVeque, *Finite Difference Methods for Differential Equations*, University of Washington, AMath 585, Winter Quarter, Seattle, Wash, USA, 2006.

[3] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[4] S. A. Hoda and H. A. Nagla, "On neural network methods for mixed boundary value problems," *International Journal of Nonlinear Science*, vol. 11, no. 3, pp. 312–316, 2011.

[5] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[6] K. S. Mc Fall and J. R. Mahan, "Investigation of weight reuse in multi-layer perceptron networks for accelerating the solution of differential equations," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 14, pp. 109–114, 2004.

[7] L. N. M. Tawfiq, *Design and training artificial neural networks for solving differential equations [Ph.D. thesis]*, University of Baghdad, College of Education-Ibn-Al-Haitham, Baghdad, Iraq, 2004.

[8] A. Malek and R. Shekari Beidokhti, "Numerical solution for high order differential equations using a hybrid neural network-Optimization method," *Applied Mathematics and Computation*, vol. 183, no. 1, pp. 260–271, 2006.

[9] H. Akca, M. H. Al-Lail, and V. Covachev, "Survey on wavelet transform and application in ODE and wavelet networks," *Advances in Dynamical Systems and Applications*, vol. 1, no. 2, pp. 129–162, 2006.

[10] K. S. Mc Fall, *An artificial neural network method for solving boundary value problems with arbitrary irregular boundaries [Ph.D. thesis]*, Georgia Institute of Technology, Atlanta, Ga, USA, 2006.

[11] A. Junaid, M. A. Z. Raja, and I. M. Qureshi, "Evolutionary computing approach for the solution of initial value problems in ordinary differential equations," *World Academy of Science, Engineering and Technology*, vol. 55, pp. 578–581, 2009.

[12] R. M. A. Zahoor, J. A. Khan, and I. M. Qureshi, "Evolutionary computation technique for solving Riccati differential equation of arbitrary order," *World Academy of Science, Engineering and Technology*, vol. 58, pp. 303–308, 2009.

[13] J. Abdul Samath, P. S. Kumar, and A. Begum, "Solution of linear electrical circuit problem using neural networks," *International Journal of Computer Applications*, vol. 2, no. 1, pp. 6–13, 2010.

[14] K. I. Ibraheem and B. M. Khalaf, "Shooting neural networks algorithm for solving boundary value problems in ODEs," *Applications and Applied Mathematics*, vol. 6, no. 11, pp. 1927–1941, 2011.

[15] K. Majidzadeh, "Inverse problem with respect to domain and artificial neural network algorithm for the solution," *Mathematical Problems in Engineering*, vol. 2011, Article ID 145608, 16 pages, 2011.

[16] Y. A. Oraibi, *Design feed forward neural networks for solving ordinary initial value problem [M.S. thesis]*, University of Baghdad, College of Education-Ibn-Al-Haitham, Baghdad, Iraq, 2011.

[17] M. H. Ali, *Design fast feed forward neural networks to solve two point boundary value problems [M.S. thesis]*, University of Baghdad, College of Education-Ibn-Al-Haitham, Baghdad, Iraq, 2012.

[18] L. N. M. Tawfiq and A. A. T. Hussein, "Design feed forward neural network to solve singular boundary value problems,"

ISRN Applied Mathematics, vol. 2013, Article ID 650467, 7 pages, 2013.

[19] I. A. Galushkin, *Neural Networks Theory*, Springer, Berlin, Germany, 2007.

[20] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*, Springer, New York, NY, USA, 1996.

[21] A. Ghaffari, H. Abdollahi, M. R. Khoshayand, I. S. Bozchalooi, A. Dadgar, and M. Rafiee-Tehrani, "Performance comparison of neural network training algorithms in modeling of bimodal drug delivery," *International Journal of Pharmaceutics*, vol. 327, no. 1-2, pp. 126–138, 2006.

[22] H. W. Rasheed, *Efficient semi-analytic technique for solving second order singular ordinary boundary value problems [M.S. thesis]*, University of Baghdad, College of Education-Ibn-Al-Haitham, Baghdad, Iraq, 2011.